**Project No: 1**

**Project Title: Delivery Time Prediction**

**Prepared by: Rose Mary Jose**

**Date: 23-06-23**

Prepared for “Thoughtware Training Private Limited”, under Guidance of its CEO Mr. Pattabhi Raman. The project will be subject to further research, modification and exclusive use of “Thoughtware Training Private Limited”

**Objective of the project**

Delivery time prediction has become crucial due to rising demands for efficient logistics and e-commerce. Accurate predictions offer benefits like enhanced customer satisfaction, optimized resource allocation, and reduced lead times. By analyzing historical delivery data, businesses can improve inventory management and demand forecasting. Real-time tracking and route optimization lead to resource efficiency and better service level agreements. Delivery time prediction finds applications in e-commerce, logistics, manufacturing, and food delivery, aiding user experience, cost reduction, and process optimization. Overall, it empowers data-driven decisions, boosts customer loyalty, and ensures timely, reliable, and eco-friendly deliveries, thus offering a competitive edge in the modern business landscape.

There are several techniques that can be used for lead time prediction, such as time series analysis, regression analysis, and machine learning algorithms like decision trees, random forests, and neural networks. The choice of technique depends on the nature of the data and the specific requirements of the problem.

The dataset taken into consideration for this project is an estimation of food delivery time because vendor lead time prediction data is not currently available.

**Dataset**

Source:  [https://www.kaggle.com/datasets/ranitsarkar01/porter-delivery-time-estimation](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Franitsarkar01%2Fporter-delivery-time-estimation)

For delivery time prediction a food delivery time dataset is considered from Kaggle. The considered dataset consists of the following columns:

market id: integer id for the market where the restaurant lies

created at: the timestamp at which the order was placed

actual delivery time: the timestamp when the order was delivered

store\_primary\_category: category for the restaurant

order protocol: integer code value for order protocol (how the order was placed le: through porter, call to restaurant, pre booked, third part etc)

total items subtotal: final price of the order

num\_distinct items: the number of distinct items in the order

actual delivery time: the timestamp when the order was delivered

store\_primary\_category: category for the restaurant

order protocol: integer code value for order protocol (how the order was placed le: through porter, call to restaurant, pre booked, third part etc)

total items subtotal: final price of the order

num\_distinct items: the number of distinct items in the order

min\_item\_price: price of the cheapest item in the order

max\_item\_price: price of the costliest item in order

total\_onshift\_partners: number of delivery partners on duty at the time order was placed

total\_busy\_partners: number of delivery partners attending to other tasks

total outstanding\_orders: total number of orders to be fulfilled at the moment

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import io  
import warnings  
warnings.filterwarnings('ignore')  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error

data = pd.read\_csv('porter\_deliverytime\_dataset.csv')  
data.head()

market\_id created\_at actual\_delivery\_time \  
0 1.0 06-02-2015 22:24 06-02-2015 23:27   
1 2.0 10-02-2015 21:49 10-02-2015 22:56   
2 3.0 22-01-2015 20:39 22-01-2015 21:09   
3 3.0 03-02-2015 21:21 03-02-2015 22:13   
4 3.0 15-02-2015 02:40 15-02-2015 03:20   
  
 store\_id store\_primary\_category order\_protocol \  
0 df263d996281d984952c07998dc54358 american 1.0   
1 f0ade77b43923b38237db569b016ba25 mexican 2.0   
2 f0ade77b43923b38237db569b016ba25 NaN 1.0   
3 f0ade77b43923b38237db569b016ba25 NaN 1.0   
4 f0ade77b43923b38237db569b016ba25 NaN 1.0

total\_items subtotal num\_distinct\_items min\_item\_price max\_item\_price \  
0 4 3441 4 557 1239   
1 1 1900 1 1400 1400   
2 1 1900 1 1900 1900   
3 6 6900 5 600 1800   
4 3 3900 3 1100 1600   
  
 total\_onshift\_partners total\_busy\_partners total\_outstanding\_orders   
0 33.0 14.0 21.0   
1 1.0 2.0 2.0   
2 1.0 0.0 0.0   
3 1.0 1.0 2.0   
4 6.0 6.0 9.0

Since the project's goal is to predict delivery time, it is obvious from the dataset that the target variable for the project is the actual delivery time. And the feature variable is the rest of the variables that must be chosen using feature selection.

**EDA**

The shape of the dataset considered is of shape: (197428,14).

The dataset's info returns details about the data frame. It contains information on the total number of columns, column labels, datatypes, the number of columns that aren't null, memory utilization, and range index. It is clear from the information that the dataset under consideration has 10 features of float64 dtype, 5 of int64 dtype, and 4 of object dtype. The presence of null values can be detected by non-null values in 7 of the 14columns. Also, the target variable's actual delivery time can be seen to be a dtype object that needs to be converted to a numerical value.

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 197428 entries, 0 to 197427  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 market\_id 196441 non-null float64  
 1 created\_at 197428 non-null object   
 2 actual\_delivery\_time 197421 non-null object   
 3 store\_id 197428 non-null object   
 4 store\_primary\_category 192668 non-null object   
 5 order\_protocol 196433 non-null float64  
 6 total\_items 197428 non-null int64   
 7 subtotal 197428 non-null int64   
 8 num\_distinct\_items 197428 non-null int64   
 9 min\_item\_price 197428 non-null int64   
 10 max\_item\_price 197428 non-null int64   
 11 total\_onshift\_partners 181166 non-null float64  
 12 total\_busy\_partners 181166 non-null float64  
 13 total\_outstanding\_orders 181166 non-null float64  
dtypes: float64(5), int64(5), object(4)  
memory usage: 21.1+ MB

To obtain the dataset's summary statistics, describe function is used. The Describe function of the numerical column provides an overview of the central tendency, dispersion, and distributional form of the dataset. The 25th, 50th, and 75th percentile values are returned, together with the count, mean, standard deviation, minimum value, and maximum value. It is obvious that there are null values and outliers from the numerical columns' summary statistics Subtotal, min item price, max item price, total on-shift partners, total busy partners, and total outstanding orders are attributes that have negative values and a broad range of values.

count mean std min 25% \  
market\_id 196441.0 2.978706 1.524867 1.0 2.0   
order\_protocol 196433.0 2.882352 1.503771 1.0 1.0   
total\_items 197428.0 3.196391 2.666546 1.0 2.0   
subtotal 197428.0 2682.331402 1823.093688 0.0 1400.0   
num\_distinct\_items 197428.0 2.670791 1.630255 1.0 1.0   
min\_item\_price 197428.0 686.218470 522.038648 -86.0 299.0   
max\_item\_price 197428.0 1159.588630 558.411377 0.0 800.0   
total\_onshift\_partners 181166.0 44.808093 34.526783 -4.0 17.0   
total\_busy\_partners 181166.0 41.739747 32.145733 -5.0 15.0   
total\_outstanding\_orders 181166.0 58.050065 52.661830 -6.0 17.0   
  
 50% 75% max   
market\_id 3.0 4.0 6.0   
order\_protocol 3.0 4.0 7.0   
total\_items 3.0 4.0 411.0   
subtotal 2200.0 3395.0 27100.0   
num\_distinct\_items 2.0 3.0 20.0   
min\_item\_price 595.0 949.0 14700.0   
max\_item\_price 1095.0 1395.0 14700.0   
total\_onshift\_partners 37.0 65.0 171.0   
total\_busy\_partners 34.0 62.0 154.0   
total\_outstanding\_orders 41.0 85.0 285.0

From the summary statistics of the categorical columns count, number of unique values, the value with highest frequency and its frequency are all given.

count unique top Freq

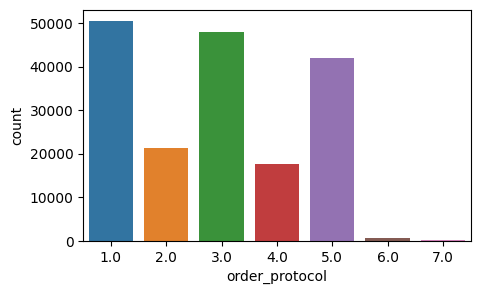
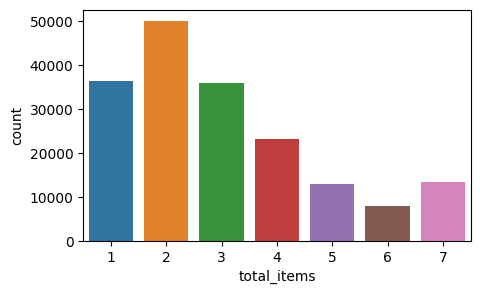
created\_at 197428 23636 11-02-2015 19:51 49  
actual\_delivery\_time 197421 23756 14-02-2015 03:21 39  
store\_id 197428 6743 d43ab110ab2489d6b9b2caa 937

394bf920f  
store\_primary\_category192668 74 american 19399

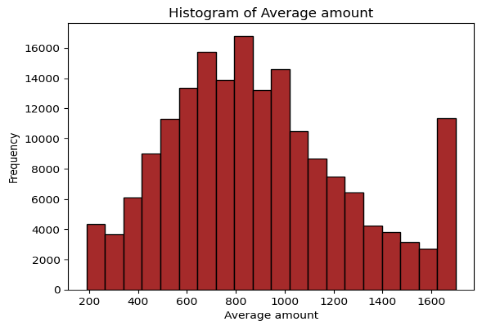
**Data visualization**

Data visualization is an effective approach for illustrating information, which makes it simpler to identify patterns, relationships, and trends in data. Visualizations offer insights that are often challenging to derive from raw data, enabling decision-making, presenting findings, and revealing hidden insights.

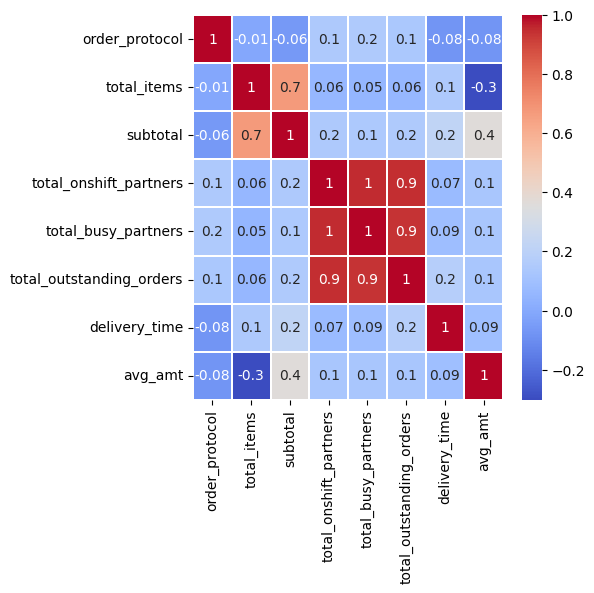
Count plot is plotted for the categorical features order protocol and total items which visually shows the number of order protocols and its count for the count plot of order protocol and total items in the dataset we considered and their count is visualized using the count plot of total items.

****  

Histograms provide valuable insights into the distribution and characteristics of a dataset. By visualizing the frequency of data values within specified bins, histograms can reveal patterns, central tendencies, dispersion, and potential outliers. From the histogram of the average amount a complete comprehension of the range of the amount of food given is derived. The mode of that feature can be seen from the peak of the histogram, and since it is close to the center, so it may be inferred that the feature has a symmetric distribution. Additionally, because the feature includes two peaks, it is bimodal.

 ****

It is visible in the delivery time histogram that the feature column has a positive skewness. Additionally, this characteristic has a bimodal distribution with values within 20 and 80 minutes.



A correlation heatmap provides a visual representation of the pairwise correlations between variables in a dataset. It is particularly useful to identify relationships and dependencies among variables. The dataset's correlation heatmap makes it possible to see how strongly all of the dataset's attributes are correlated. While negative values signify a negative correlation, positive values show a positive connection. The target variable delivery time doesn't appear to have a strong correlation with any of the attributes. But multicollinearity is present. In order to treat multicollinearity, the columns which show strong correlation between feature variables are eliminated because these variables that exhibit multicollinearity don't significantly affect the prediction of delivery time.

**Data Preparation**

**Detecting and handling missing values**: To ensure accuracy, reliability, and robustness of analyses and models, data preparation is essential for detecting and handling missing values and outliers. Due to the fact that missing values result from errors or inadequate recording and outliers represent anomalies or exceptional circumstances, these problems may result in biased conclusions and incorrect predictions. For the sake of maintaining data integrity and arriving with suitable conclusion, these issues must be regularly addressed.

For detecting the missing data is to use Python functions like isnull() and sum(). The isnull().sum() function helps to quickly figure out the amount of data missing from each column. In this step, one can observe which columns have missing values and deal with them.

data.isnull().sum()

market\_id 987  
created\_at 0  
actual\_delivery\_time 7  
store\_id 0  
store\_primary\_category 4760  
order\_protocol 995  
total\_items 0  
subtotal 0  
num\_distinct\_items 0  
min\_item\_price 0  
max\_item\_price 0  
total\_onshift\_partners 16262  
total\_busy\_partners 16262  
total\_outstanding\_orders 16262  
dtype: int64

It is observed from the missing values that columns like "market\_id," "store\_primary\_category," "store\_id," and "num\_distinct\_items" can be dropped since they don't have much significance in the model building. Rest missing values can be dropped using the dropna function, where the rows with missing values are dropped. The shape of the dataset after dropping missing values is (180242,10).

df = data.drop(['market\_id','store\_primary\_category','store\_id','num\_distinct\_items'],axis = 1)

df.dropna(axis = 0,how = 'any', inplace = True)  
df.shape

(180242, 10)

**Creating new columns from existing columns:** From info it is evident that target variable is of dtype object. So to calculate the delivery time, actual\_delivery\_time and created\_at columns are converted into datetime dtype and then they are subtracted from each other. Thus the target variable is created and of dtype float64. From columns max\_item\_price and min\_item\_price new column is created called avg\_amt of dtype float64.

df['actual\_delivery\_time'] = pd.to\_datetime(df['actual\_delivery\_time'])

df['created\_at'] = pd.to\_datetime(df['created\_at'])

#Delivery time  
df['delivery\_time'] = df['actual\_delivery\_time'] - df['created\_at']

#Delivery time in minutes  
from datetime import timedelta  
df['delivery\_time'] = df['delivery\_time'] /timedelta(minutes=1)  
df.drop(['actual\_delivery\_time','created\_at'],inplace=True,axis=1)  
  
  
df['avg\_amt'] = (df['min\_item\_price'] + df['max\_item\_price']) / 2  
df.drop(['max\_item\_price','min\_item\_price'],inplace=True,axis=1)

**Detecting and handling duplicated values:** Duplicated values are identical or repeated entries in a dataset, resulting from errors, system glitches, or unintentional repetitions. Identifying and handling duplicates is crucial in data preprocessing to maintain integrity and ensure accurate analyses. Functions like duplicated() in Python can detect duplicates and mark subsequent occurrences as duplicates. The value count of the duplicated values is given. From this it can be concluded that there are about 33 dupliacted values

df.duplicated().value\_counts()

False 180209  
True 33  
dtype: int64

Addressing duplicates ensures data consistency, reliability, and prevents bias, contributing to data quality and trustworthiness. Here the duplicated values are dropped and keep the last occurence of each duplicated set. The shape of the dataset after dropping dupliacted value is (180209,8).

# Keep the last occurrence of duplicates  
df = df.drop\_duplicates()

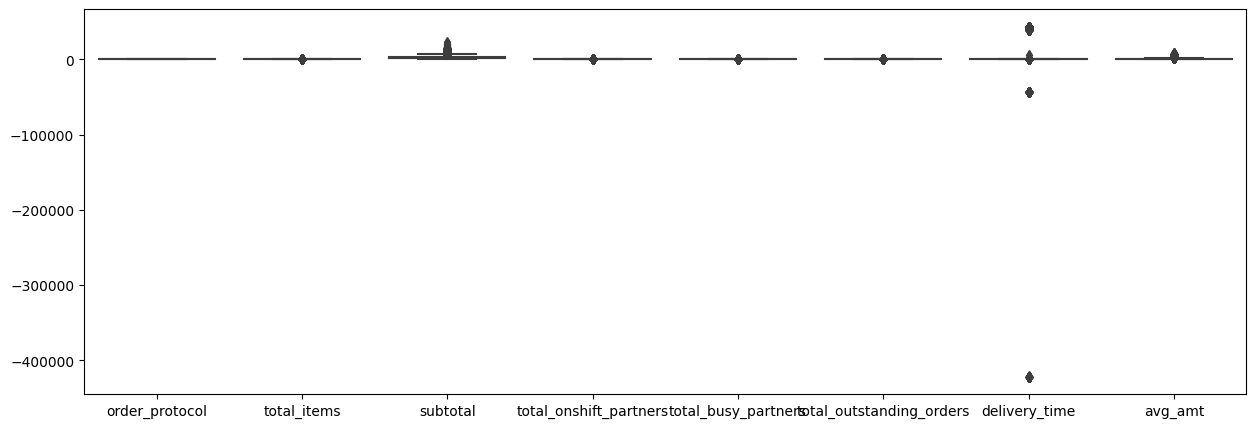
df.duplicated().sum()

df.shape

0

(180209, 8)

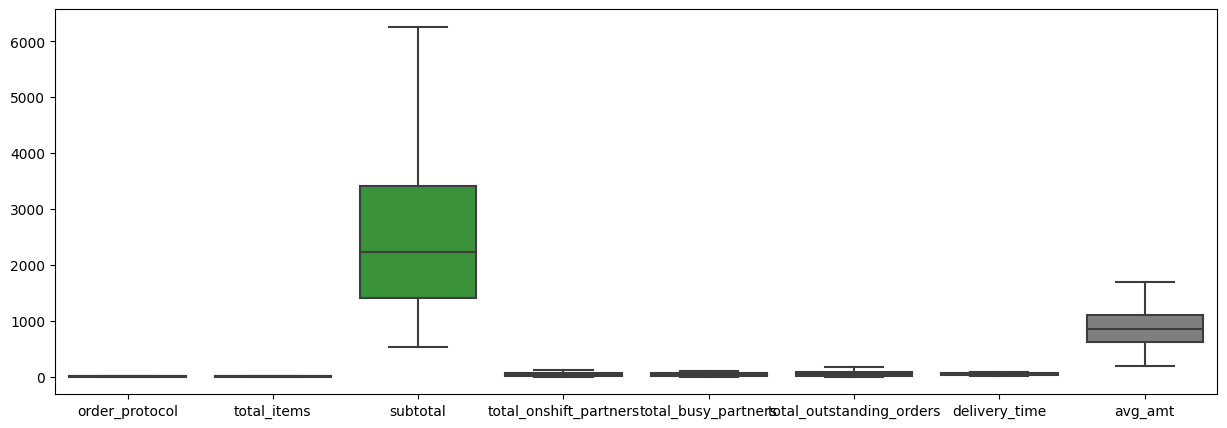
**Detecting and handling outliers**: Identifying outliers is vital to maintain the integrity of data distributions and to prevent them from disproportionately affecting analysis. Visualizations, particularly boxplots, are powerful tools for detecting outliers. A boxplot provides a graphical representation of the data's spread, central tendency, and potential outliers. Outliers manifest as individual points beyond the "whiskers" of the boxplot. Boxplot of the numerical columns are plotted for detecting the presence of outliers and the image is given below.



From the boxplot, it is clear that outliers are present. Winsorization method is used to handle the outliers. Winsorization replaces extreme values with the nearest "normal" value. For example, the 1% of values on each side of the distribution are replaced with the 1st and 99th percentile values, respectively. Here the columns that contain outliers are selected and applied the winsorization technique. 5% of values that are present on each side is replaced with 5th and 95th percentile values.

from scipy.stats.mstats import winsorize  
# handling outliers using winsorize method  
for col in df[['delivery\_time','avg\_amt','subtotal','total\_items','total\_onshift\_partners','total\_busy\_partners','total\_outstanding\_orders']]:  
 df[col] = winsorize(df[col],limits=[0.01,0.05])

Boxplot plotted after outlier removal is given below. From the boxplot it is observable that the outliers are handled properly.



**Standardizing data using Min-Max Scaler**: Min-Max Scaling enhances model performance by preventing scale-related biases, ensuring that all features contribute equitably to the predictions. It's a valuable preprocessing step that contributes to more reliable and accurate predictions.

Using Min-Max Scaling in the context of delivery time prediction involves transforming numerical features to a common scale, typically between 0 and 1. This scaling technique maintains the relative relationships between different features while preventing any particular feature from dominating the analysis due to its larger magnitude. Applying Min-Max Scaling before model training ensures that no particular feature's scale unfairly influences the prediction process. Models like Linear Regression and K-Neighbors Regression, which are sensitive to feature scales, benefit from Min-Max Scaling. Random Forest Regression, which is less affected by scale differences, may not always require this scaling. For each feature, the Min-Max Scaling process subtracts the minimum value of that feature and then divides by the range (difference between maximum and minimum values). This operation transforms the values to a standardized range while preserving their proportionality.

from sklearn.preprocessing import MinMaxScaler  
ms=MinMaxScaler()  
X = ms.fit\_transform(X)

**Model Selection**

Model selection is a crucial process in machine learning that involves choosing the most appropriate algorithm or model to solve a specific problem. In the case of delivery time prediction, where the goal is to estimate the time, it takes for deliveries to reach their destinations, model selection plays a pivotal role in achieving accurate and reliable results. Supervised learning is a subset of machine learning where the algorithm learns from labeled training data, making predictions based on input features and their corresponding target labels. In our scenario, supervised learning involves training models on historical delivery data, with features like distance, time of day, and traffic conditions, to predict delivery times.

To tackle this challenge, three distinct supervised learning models are employed: Linear Regression, Random Forest, and K-Nearest Neighbors Regression. Linear Regression is a straightforward yet powerful method that establishes a linear relationship between input features and the target variable. It is especially suitable when there is a clear correlation between factors like distance and delivery time. Random Forest, a versatile ensemble technique, constructs multiple decision trees to capture complex relationships in the data. It excels in handling nonlinear patterns and interactions among variables, making it well-suited for scenarios where various factors contribute to delivery time variations. K-Nearest Neighbors Regression focuses on the proximity of data points. It identifies similar historical delivery instances and predicts the delivery time based on the average of their outcomes. This method is effective when delivery times are influenced by cases that share similar characteristics. The model selection process involves training each of these models on a subset of the data and evaluating their performance using metrics such as Mean Squared Error or R-squared on a validation dataset. The model that demonstrates the best predictive accuracy and generalization is chosen for deployment.

**Propose Model Evaluation Methods**

Model evaluation methods are essential to assess the performance of machine learning models. In the context of delivery time prediction, where accuracy is crucial, several evaluation metrics can be employed. One such metric is the Root Mean Squared Error (RMSE). Evaluation metrics allow us to quantitatively measure how well a model's predictions align with the actual delivery times. RMSE is particularly important as it calculates the average difference between predicted and actual delivery times while penalizing larger errors more heavily. This is crucial in a domain where accurate delivery time estimates are vital for customer satisfaction and operational efficiency.

RMSE, specifically, measures the square root of the average of squared errors between predicted and actual values. Its use provides a comprehensive understanding of the model's overall performance, taking into account both small and large errors. Lower RMSE values indicate better predictive accuracy, making it a reliable tool for model comparison and selection in the context of delivery time prediction.

**Split training/testing final dataset**

Splitting a dataset into three subsets—training, validation, and testing—is a fundamental practice in machine learning to assess model performance, prevent overfitting, and ensure generalization. In this scenario, a 70:30 split using the train\_test\_split function creates distinct training and testing sets. An additional 10% slice is allocated for the validation dataset, chosen via the sample function. This division enables models to learn patterns from the training set, tune hyperparameters on the validation set, and ultimately evaluate performance on the untouched testing set. Such a split ensures a balanced approach to model development, refinement, and assessment, contributing to robust and accurate predictions in the delivery time prediction context.

## feature variables  
X = df.drop(['delivery\_time'],axis = 1)  
## target varibale  
y = df['delivery\_time']

Xtrain,Xtest,ytrain,ytest = train\_test\_split(X,y,test\_size = 0.3,random\_state = 42)

#### #splitting data for validation

sample\_df = df.sample(frac = 0.01,random\_state = 1)

df = df.drop(sample\_df.index)

**Models**

Linear Regression: Linear Regression would analyze the relationship between the selected predictor variables (order\_protocol, total\_items, avg\_amt) and the target variable (delivery\_time). It assumes a linear connection, estimating how changes in these predictors contribute to changes in delivery time. This model could reveal how an increase or decrease in any of the feature variable might affect delivery time. However, it might oversimplify complex interactions present in the data.

**Random Forest Regression**: Random Forest would delve into the interactions between these features and delivery time. By creating numerous decision trees and aggregating their results, it can capture intricate relationships, accounting for nonlinearities and interdependencies. This model could uncover how order\_protocol, total\_items, and avg\_amt collectively impact delivery time, providing a richer understanding.

**K-Neighbors Regression**: K-Neighbors Regression would focus on the similarity between instances in the dataset. It would predict delivery time based on the average of nearby cases in terms of order\_protocol, total\_items, and avg\_amt. This approach is valuable when historical deliveries with similar characteristics influence present delivery times. It can offer insights into local patterns, allowing for nuanced predictions.

Rmse = pd.DataFrame({'model':["Random forest","Linear regression","K-neighbour"],'Rmse\_train':[rfr\_train\_rmse,lr\_train\_rmse,KNN\_train\_rmse],'Rmse\_test':[rfr\_test\_rmse,lr\_test\_rmse,KNN\_test\_rmse]})  
Rmse

Here, for all three models the training and testing dataset are considered and model is fitted and prediction of both Xtrain and Xtest is done and their RMSE values are calculated. A data frame with the RMSE of both train and test data is displayed below.

model Rmse\_train Rmse\_test  
0 Random forest 16.096519 16.129621  
1 Linear regression 16.046049 16.077995  
2 K-neighbour 15.910971 17.255891

**Model’s Summary**

In our delivery time prediction analysis, we employed three distinct models: Linear Regression, Random Forest Regression, and K-Neighbors Regression. Evaluating these models based on their Root Mean Squared Error (RMSE) values provides insights into their predictive accuracy.

Linear Regression yielded RMSE values of 16.09 for the training data and 16.12 for the test data. This model captured linear relationships between predictors (order\_protocol, total\_items, avg\_amt) and delivery time. However, it seems to slightly overfit, as the test RMSE is slightly higher. Random Forest Regression demonstrated a training RMSE of 16.04 and a test RMSE of 16.077. By aggregating multiple decision trees, it effectively modeled complex interactions and non-linear patterns, achieving similar performance on both training and test data. K-Neighbors Regression, with a training RMSE of 15.91 and a test RMSE of 17.25, excelled in capturing local patterns. It predicted delivery time based on the average of nearby instances with similar attributes, though it appears to exhibit some overfitting as evidenced by the higher test RMSE.

In summary, all three models exhibit reasonable predictive abilities, with Linear Regression providing a simpler interpretation of linear trends, Random Forest Regression capturing intricate relationships, and K-Neighbors Regression focusing on local patterns. While Random Forest and Linear Regression showcase more consistent performance, K-Neighbors Regression might benefit from further tuning to enhance its generalization capabilities. Further analysis and fine-tuning are recommended to select the optimal model for accurate delivery time predictions.

**Model Validation**

For validating the model, pre-processing of the valid data is done. Since the data is split after handling missing values, outliers and duplicate values, features and target variable are separated from the sample data and Min-Max scaler is used for standardizing data in the range of 0 and 1. Then linear regression and random forest regression models are used for data validation since they are the model that showed low RMSE for both training and testing data. The RMSE score after prediction are:

RMSE = pd.DataFrame({'model':["Random forest","Linear regression"],'Rmse':[rmse1,rmse2]})  
RMSE

model Rmse  
0 Random forest 16.645727  
1 Linear regression 16.560437

A data frame with the actual and the predicted values using both linear regression and random forest regression is created and the head of that data frame is given below.

delivery\_time randomforest\_pred linearreg\_pred  
33907 50.0 45.598357 46.748352  
13397 52.0 51.449933 53.392758  
38703 50.0 47.520226 48.385971  
78585 47.0 44.573856 42.111225  
20085 47.0 56.635044 55.475620  
... ... ... ...  
162604 64.0 56.635044 57.893949  
152735 69.0 44.052121 45.026860  
8025 48.0 51.449933 50.398176  
46136 37.0 45.557772 43.931119  
164454 42.0 47.666963 52.932187

**Conclusion**

In this project focused on delivery time prediction, we embarked on a comprehensive journey to develop accurate and reliable models. Our analysis encompassed crucial steps, including data preprocessing, feature selection, model selection, and evaluation. Three prominent supervised learning models, namely Linear Regression, Random Forest Regression, and K-Neighbors Regression, were employed to predict delivery times based on selected features.

The project successfully developed robust models for delivery time prediction, demonstrating the power of machine learning. Linear Regression and Random Forest Regression demonstrated commendable predictive abilities, with Linear Regression showing a validation RMSE of 16.64 and Random Forest Regression showing a slightly improved RMSE of 16.56. These findings are crucial for logistics planning, resource allocation, customer satisfaction, and operational efficiency. The project's efficacy is demonstrated through meticulous data preparation, feature engineering, model selection, and validation.

**References**

<https://www.kaggle.com/datasets/ranitsarkar01/porter-delivery-time-estimation>